**Задание 1**

#include <iostream>

#include <cstdlib>

#include "Windows.h"

using namespace std;

void getProcessPriority(HANDLE hp); //приоритетный класс процесса

void getThreadPriority(HANDLE ht);

void getAffinityMask(HANDLE hp, HANDLE ht);

int main()

{

HANDLE hp = GetCurrentProcess();

HANDLE ht = GetCurrentThread();

DWORD pid = GetCurrentProcessId();

DWORD tid = GetCurrentThreadId();

cout << "OS05\_01\n\n";

cout << "PID: " << pid << endl << "TID: " << tid << endl;

getProcessPriority(hp);

getThreadPriority(ht);

getAffinityMask(hp, ht);

system("pause");

}

void getProcessPriority(HANDLE hp) {

DWORD prty = GetPriorityClass(hp);

switch (prty) {

case IDLE\_PRIORITY\_CLASS: cout << "ProcessPriority: IDLE\_PRIORITY\_CLASS\n"; break;

case BELOW\_NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: BELOW\_NORMAL\_PRIORITY\_CLASS\n"; break;

case NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: NORMAL\_PRIORITY\_CLASS\n"; break;

case ABOVE\_NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: ABOVE\_NORMAL\_PRIORITY\_CLASS\n"; break;

case HIGH\_PRIORITY\_CLASS: cout << "ProcessPriority: HIGH\_PRIORITY\_CLASS\n"; break;

case REALTIME\_PRIORITY\_CLASS: cout << "ProcessPriority: REALTIME\_PRIORITY\_CLASS\n"; break;

default: cout << "?\n\n"; break;

}

}

void getThreadPriority(HANDLE ht) {

DWORD prty = GetPriorityClass(ht);

cout << "ThreadPriority: " << GetThreadPriority(ht) << " ";

switch (prty) {

case THREAD\_PRIORITY\_LOWEST: cout << "THREAD\_PRIORITY\_LOWEST\n" ; break;

case THREAD\_PRIORITY\_BELOW\_NORMAL: cout << "THREAD\_PRIORITY\_BELOW\_NORMAL\n" ; break;

case THREAD\_PRIORITY\_NORMAL: cout << "THREAD\_PRIORITY\_NORMAL\n" ; break;

case THREAD\_PRIORITY\_ABOVE\_NORMAL: cout << "THREAD\_PRIORITY\_ABOVE\_NORMAL\n" ; break;

case THREAD\_PRIORITY\_HIGHEST: cout << "THREAD\_PRIORITY\_HIGHEST\n" ; break;

case THREAD\_PRIORITY\_IDLE: cout << "THREAD\_PRIORITY\_IDLE\n" ; break;

default: cout << "?\n\n"; break;

}

}

void getAffinityMask(HANDLE hp, HANDLE ht) {

DWORD pa = NULL, sa = NULL, icpu = -1;

char buf[10];

if (!GetProcessAffinityMask(hp, &pa, &sa)) {

throw "GetProcessAffinityMask";

}

\_itoa\_s(pa, buf, 2);

cout << "Process affinity Mask: " << buf << endl;

cout << "Process affinity Mask: " << hex << pa << endl;

\_itoa\_s(sa, buf, 2);

cout << "System affinity Mask: " << buf << endl;

icpu = SetThreadIdealProcessor(ht, MAXIMUM\_PROCESSORS);

cout << "Thread IdealProcessor: " << dec << icpu<< endl;

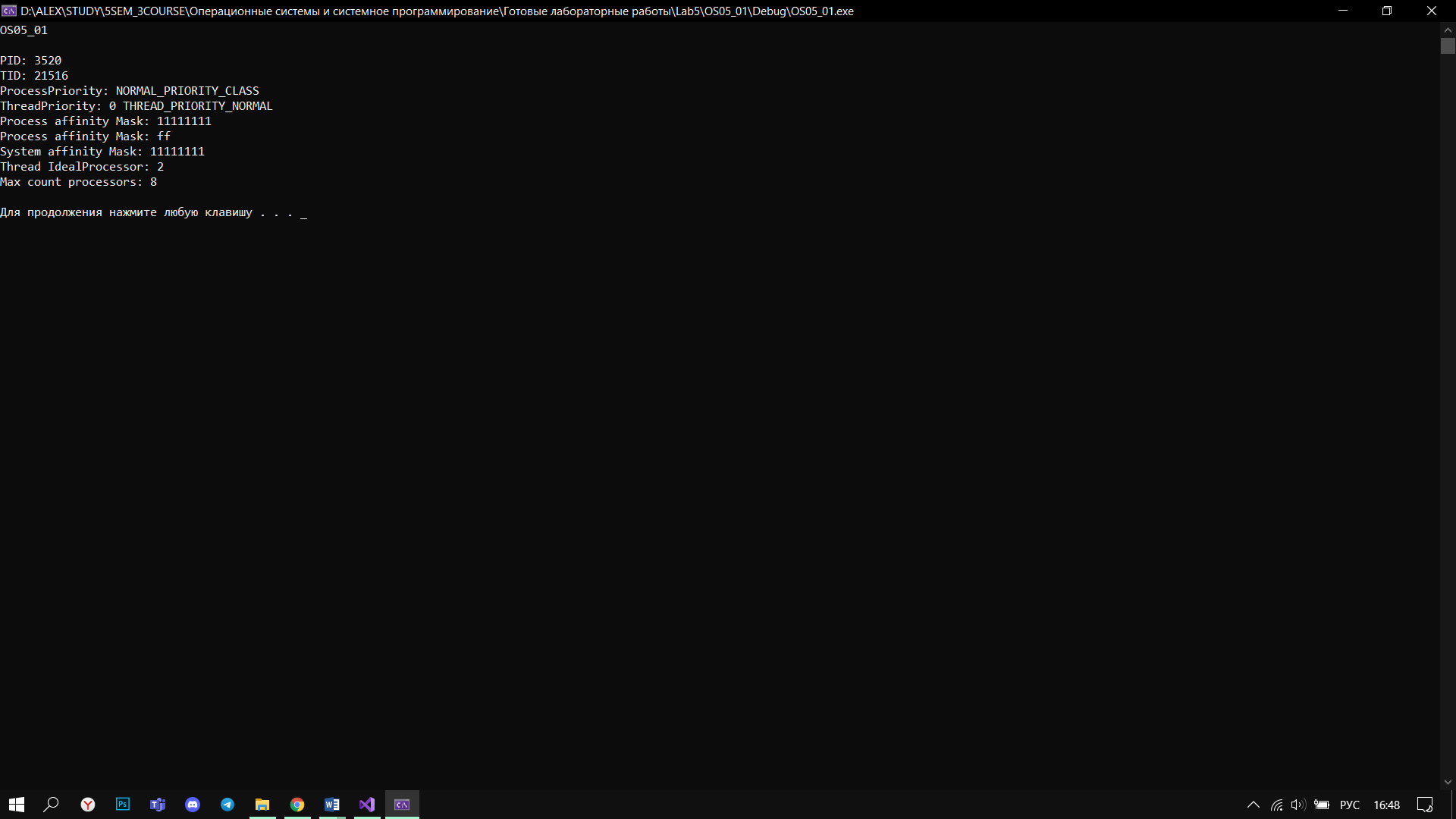
SYSTEM\_INFO sys\_info;

GetSystemInfo(&sys\_info);

cout << "Max count processors: " << sys\_info.dwNumberOfProcessors << endl << endl;

}

**OS05\_01**



**Задание 2**

#include <iostream>

#include <cstdlib>

#include "Windows.h"

using namespace std;

int main(int argc, char\* argv[])

{

try {

if (argc == 4) {

char buf[10];

int p1 = atoi(argv[1]);

int p2 = atoi(argv[2]);

int p3 = atoi(argv[3]);

if (!SetProcessAffinityMask(GetCurrentProcess(), p1)) {

throw "SetProcessAffinityMask";

}

\_itoa\_s(p1, buf, 2);

cout << "processAffinityMask: " << buf << endl;

cout << "processPriorityClass1: " << p2 << endl;

cout << "processPriorityClass2: " << p3 << endl;

LPCWSTR an1 = L"OS05\_02x.exe";

LPCWSTR an2 = L"OS05\_02x.exe";

STARTUPINFO si1, si2;

PROCESS\_INFORMATION pi1, pi2;

ZeroMemory(&si1, sizeof(STARTUPINFO)); ZeroMemory(&si2, sizeof(STARTUPINFO));

si1.cb = sizeof(STARTUPINFO); si2.cb = sizeof(STARTUPINFO);

if (CreateProcess(an1, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | p2, NULL, NULL, &si1, &pi1)) {

cout << "--Process OS05\_02 1 created\n";

}

else {

cout << "--Process OS05\_02 1 not created\n";

}

if (CreateProcess(an2, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | p3, NULL, NULL, &si2, &pi2)) {

cout << "--Process OS05\_02 2 created\n";

}

else {

cout << "--Process OS05\_02 2 not created\n";

}

WaitForSingleObject(pi1.hProcess, INFINITE);

WaitForSingleObject(pi2.hProcess, INFINITE);

CloseHandle(pi1.hProcess);

CloseHandle(pi2.hProcess);

}

else {

cout << "No arguments" << endl;

}

}

catch (string err) {

cout << err << endl;

}

system("pause");

}

**OS05\_02**

#include <iostream>

#include "Windows.h"

using namespace std;

void getProcessPriority(HANDLE hp); //приоритетный класс процесса

void getThreadPriority(HANDLE ht);

void getAffinityMask(HANDLE hp, HANDLE ht);

int main()

{

HANDLE hp = GetCurrentProcess();

HANDLE ht = GetCurrentThread();

DWORD pid = GetCurrentProcessId();

DWORD tid = GetCurrentThreadId();

for (int i = 0; i < 1000000; i++) {

if (i % 1000 == 0) {

cout << "Iteration: " << i << endl;

cout << "PID: " << pid << endl << "TID: " << tid << endl;

getProcessPriority(hp);

getThreadPriority(ht);

getAffinityMask(hp, ht);

Sleep(200);

}

}

system("pause");

}

void getProcessPriority(HANDLE hp) {

DWORD prty = GetPriorityClass(hp);

cout << prty << endl;

switch (prty) {

case IDLE\_PRIORITY\_CLASS: cout << "ProcessPriority: IDLE\_PRIORITY\_CLASS\n"; break;

case BELOW\_NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: BELOW\_NORMAL\_PRIORITY\_CLASS\n"; break;

case NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: NORMAL\_PRIORITY\_CLASS\n"; break;

case ABOVE\_NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: ABOVE\_NORMAL\_PRIORITY\_CLASS\n"; break;

case HIGH\_PRIORITY\_CLASS: cout << "ProcessPriority: HIGH\_PRIORITY\_CLASS\n"; break;

case REALTIME\_PRIORITY\_CLASS: cout << "ProcessPriority: REALTIME\_PRIORITY\_CLASS\n"; break;

default: cout << "?\n\n"; break;

}

}

void getThreadPriority(HANDLE ht) {

DWORD prty = GetPriorityClass(ht);

cout << "ThreadPriority: " << GetThreadPriority(ht) << endl;

}

void getAffinityMask(HANDLE hp, HANDLE ht) {

DWORD icpu = -1;

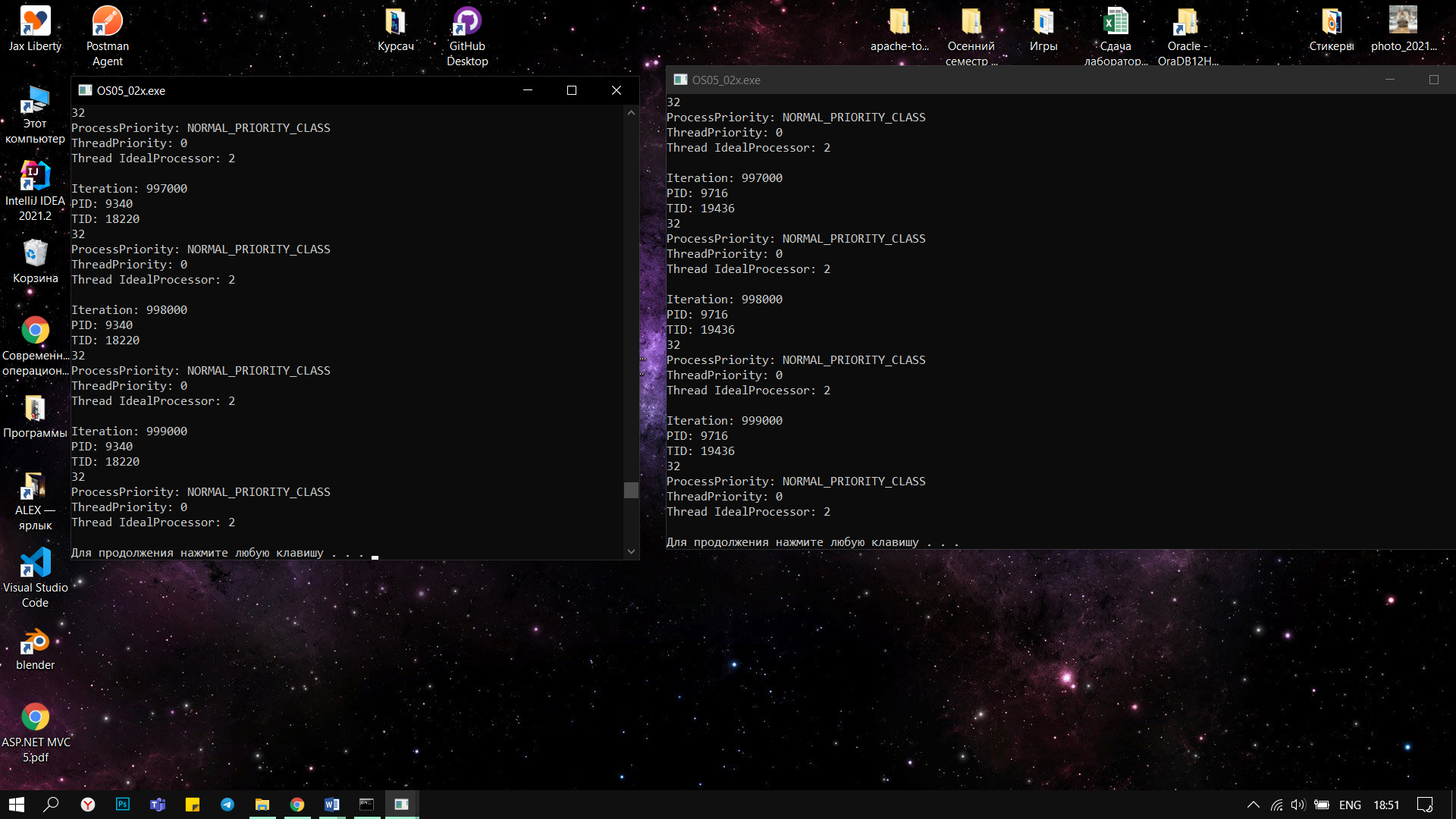
icpu = SetThreadIdealProcessor(ht, MAXIMUM\_PROCESSORS);

cout << "Thread IdealProcessor: " << dec << icpu << endl << endl;

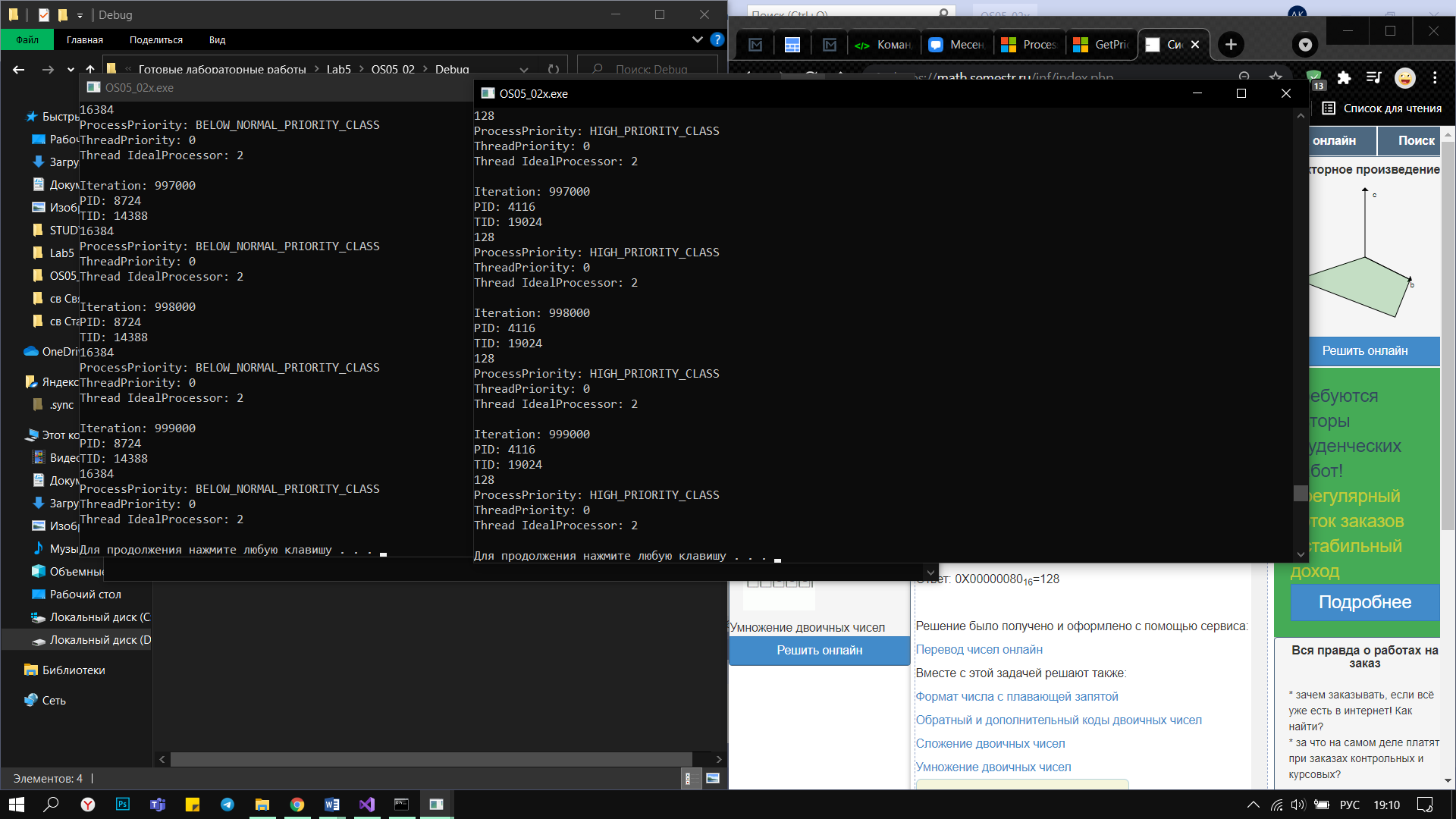
}

**OS05\_02x**

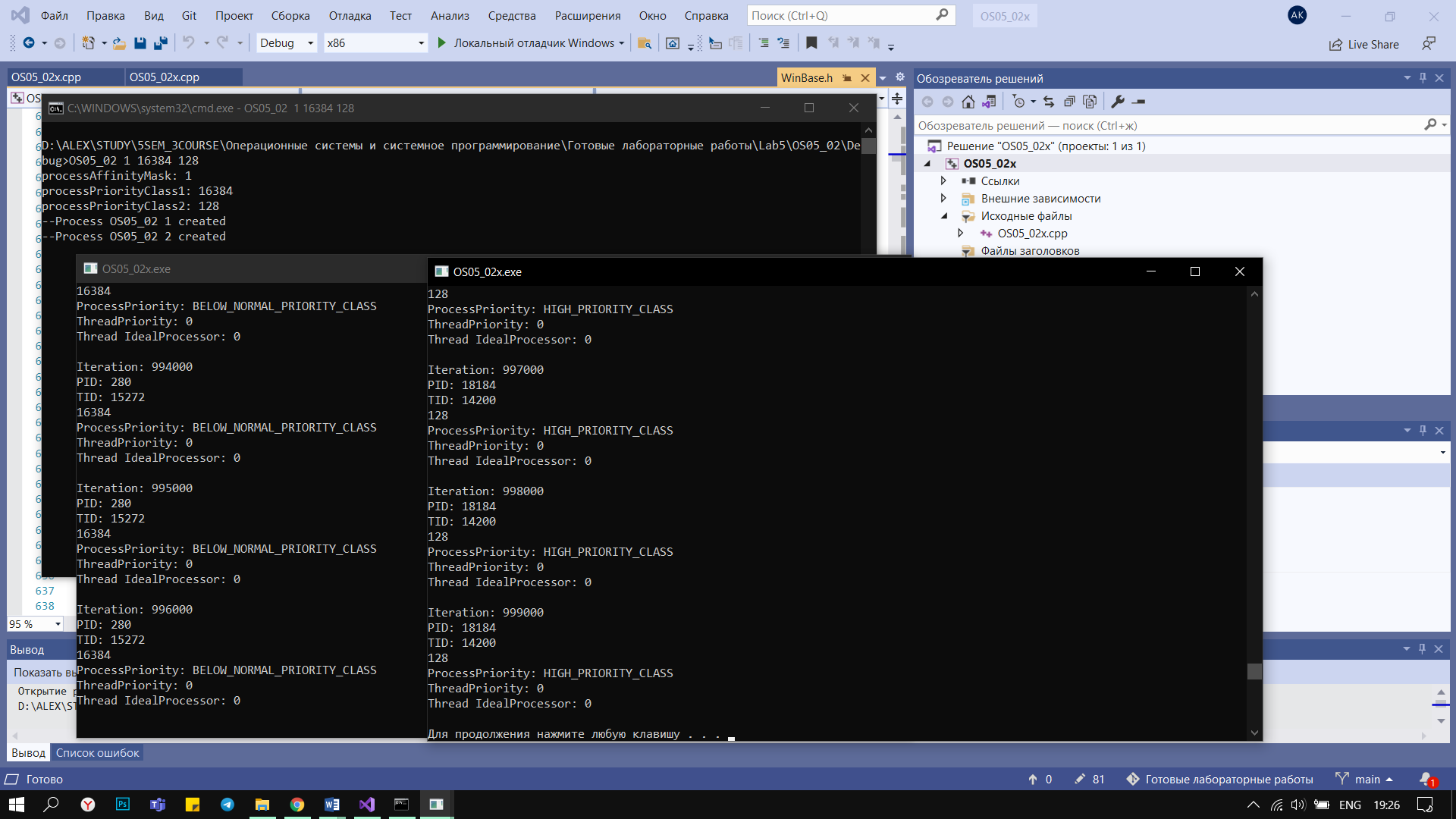
**OS05\_02 255 0 0**



**OS05\_02 255 16384 128**



**OS05\_02 1 16384 128**



**Задание 3**

#include <iostream>

#include <cstdlib>

#include "Windows.h"

using namespace std;

HANDLE createThread(int threadPriority);

DWORD WINAPI OS05\_03();

void getProcessPriority(HANDLE hp);

void getThreadPriority(HANDLE ht);

void getAffinityMask(HANDLE hp, HANDLE ht);

int main(int argc, char\* argv[])

{

if (argc == 5) {

char buf[10];

int p1 = atoi(argv[1]);

int p2 = atoi(argv[2]);

int p3 = atoi(argv[3]);

int p4 = atoi(argv[4]);

\_itoa\_s(p1, buf, 2);

cout << "processAffinityMask: " << buf << endl;

cout << "processPriorityClass: " << p2 << endl;

cout << "threadPriority1: " << p3 << endl;

cout << "threadPriority2: " << p4 << endl;

HANDLE h = GetCurrentProcess();

SetPriorityClass(h, p2);

SetProcessAffinityMask(h, p1);

HANDLE thread1 = createThread(p3);

HANDLE thread2 = createThread(p4);

ResumeThread(thread1);

ResumeThread(thread2);

HANDLE threads[] = { thread1, thread2 };

WaitForMultipleObjects(2, threads, TRUE, INFINITY);

CloseHandle(thread1);

CloseHandle(thread2);

}

else {

cout << "No arguments" << endl;

}

system("pause");

}

HANDLE createThread(int threadPriority) {

DWORD ChildId = NULL;

HANDLE h = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)OS05\_03, NULL, CREATE\_SUSPENDED, &ChildId);

SetThreadPriority(h, threadPriority);

return h;

}

DWORD WINAPI OS05\_03() {

HANDLE hp = GetCurrentProcess();

HANDLE ht = GetCurrentThread();

DWORD pid = GetCurrentProcessId();

DWORD tid = GetCurrentThreadId();

for (int i = 0; i < 1000000; i++) {

if (i % 1000 == 0) {

cout << "Iteration: " << i << endl;

cout << "PID: " << pid << endl << "TID: " << tid << endl;

getProcessPriority(hp);

getThreadPriority(ht);

getAffinityMask(hp, ht);

Sleep(200);

}

}

system("pause");

return 0;

}

void getProcessPriority(HANDLE hp) {

DWORD prty = GetPriorityClass(hp);

cout << prty << endl;

switch (prty) {

case IDLE\_PRIORITY\_CLASS: cout << "ProcessPriority: IDLE\_PRIORITY\_CLASS\n"; break;

case BELOW\_NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: BELOW\_NORMAL\_PRIORITY\_CLASS\n"; break;

case NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: NORMAL\_PRIORITY\_CLASS\n"; break;

case ABOVE\_NORMAL\_PRIORITY\_CLASS: cout << "ProcessPriority: ABOVE\_NORMAL\_PRIORITY\_CLASS\n"; break;

case HIGH\_PRIORITY\_CLASS: cout << "ProcessPriority: HIGH\_PRIORITY\_CLASS\n"; break;

case REALTIME\_PRIORITY\_CLASS: cout << "ProcessPriority: REALTIME\_PRIORITY\_CLASS\n"; break;

default: cout << "?\n\n"; break;

}

}

void getThreadPriority(HANDLE ht) {

DWORD prty = GetPriorityClass(ht);

cout << "ThreadPriority: " << GetThreadPriority(ht) << endl;

}

void getAffinityMask(HANDLE hp, HANDLE ht) {

DWORD icpu = -1;

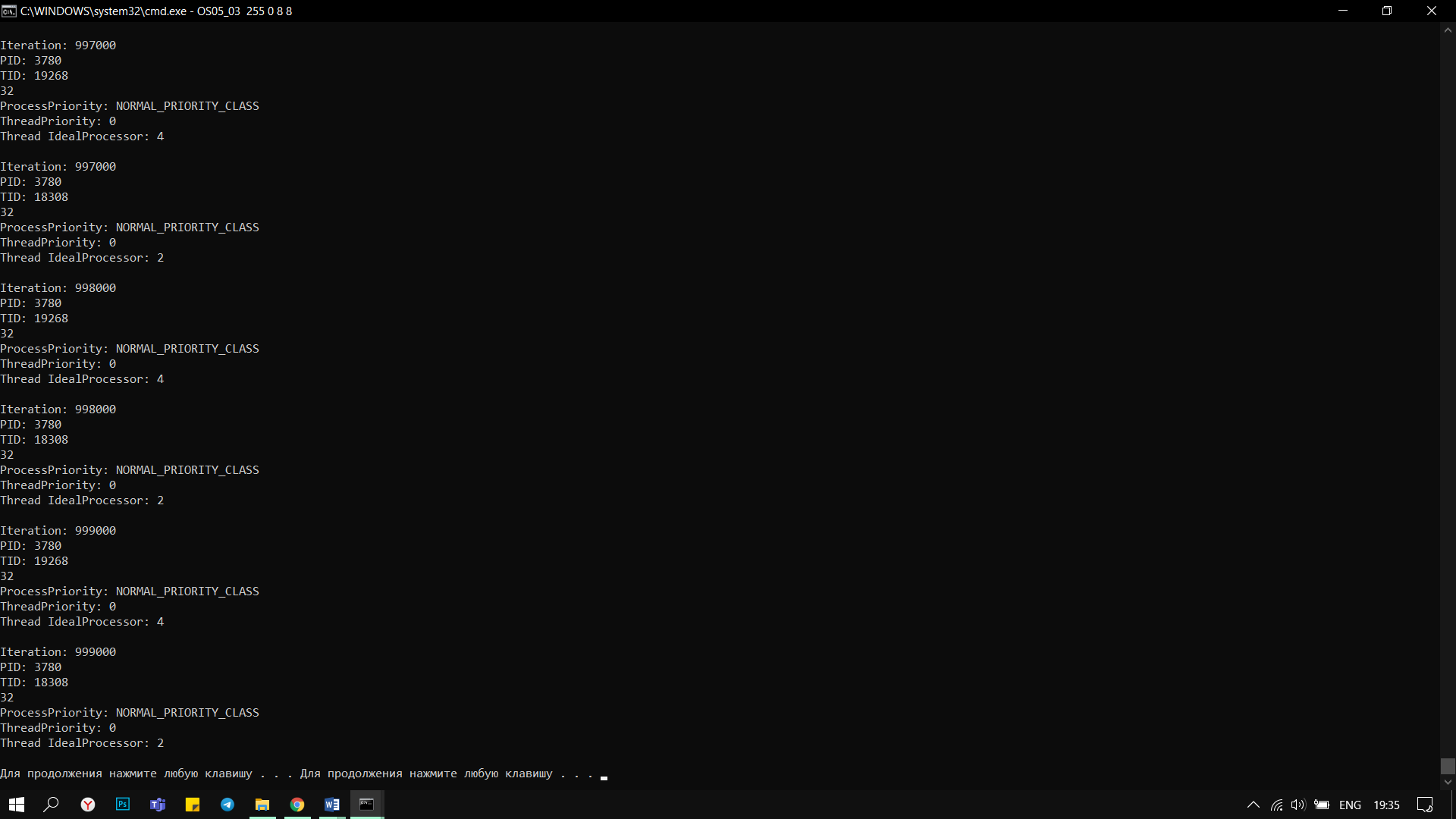
icpu = SetThreadIdealProcessor(ht, MAXIMUM\_PROCESSORS);

cout << "Thread IdealProcessor: " << dec << icpu << endl << endl;

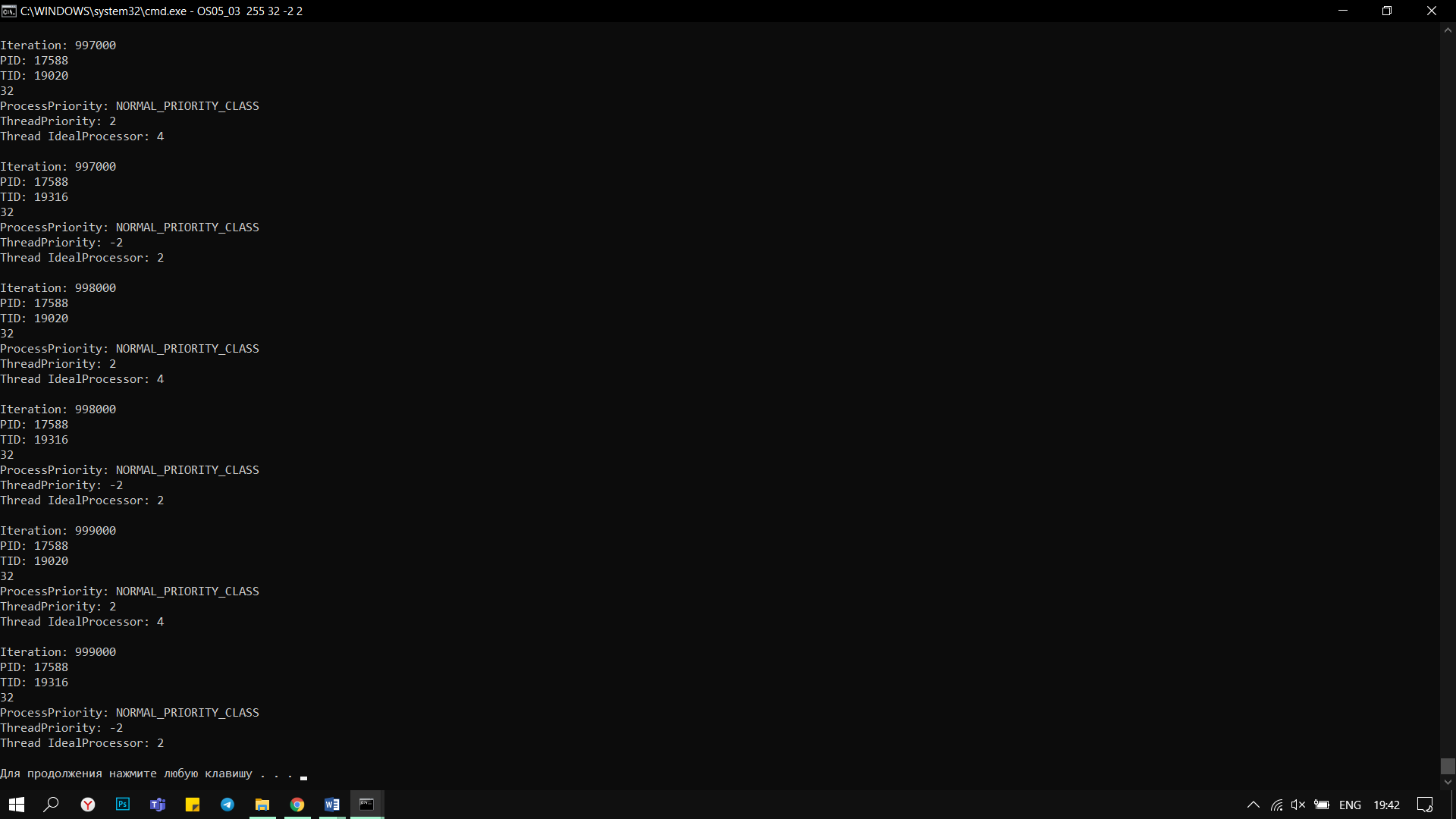
}

**OS05\_03**

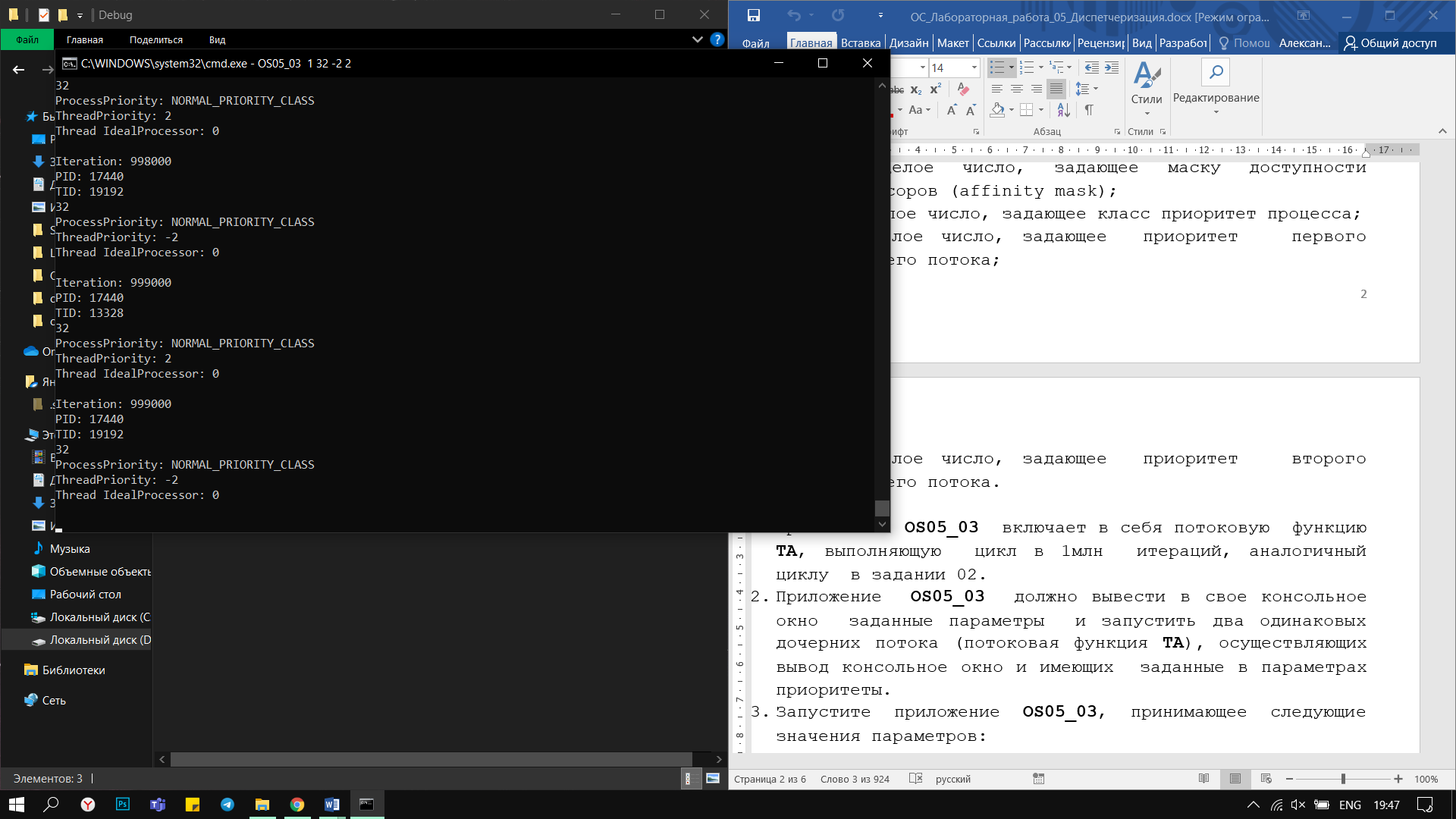
**OS05\_03 255 32 1 1**



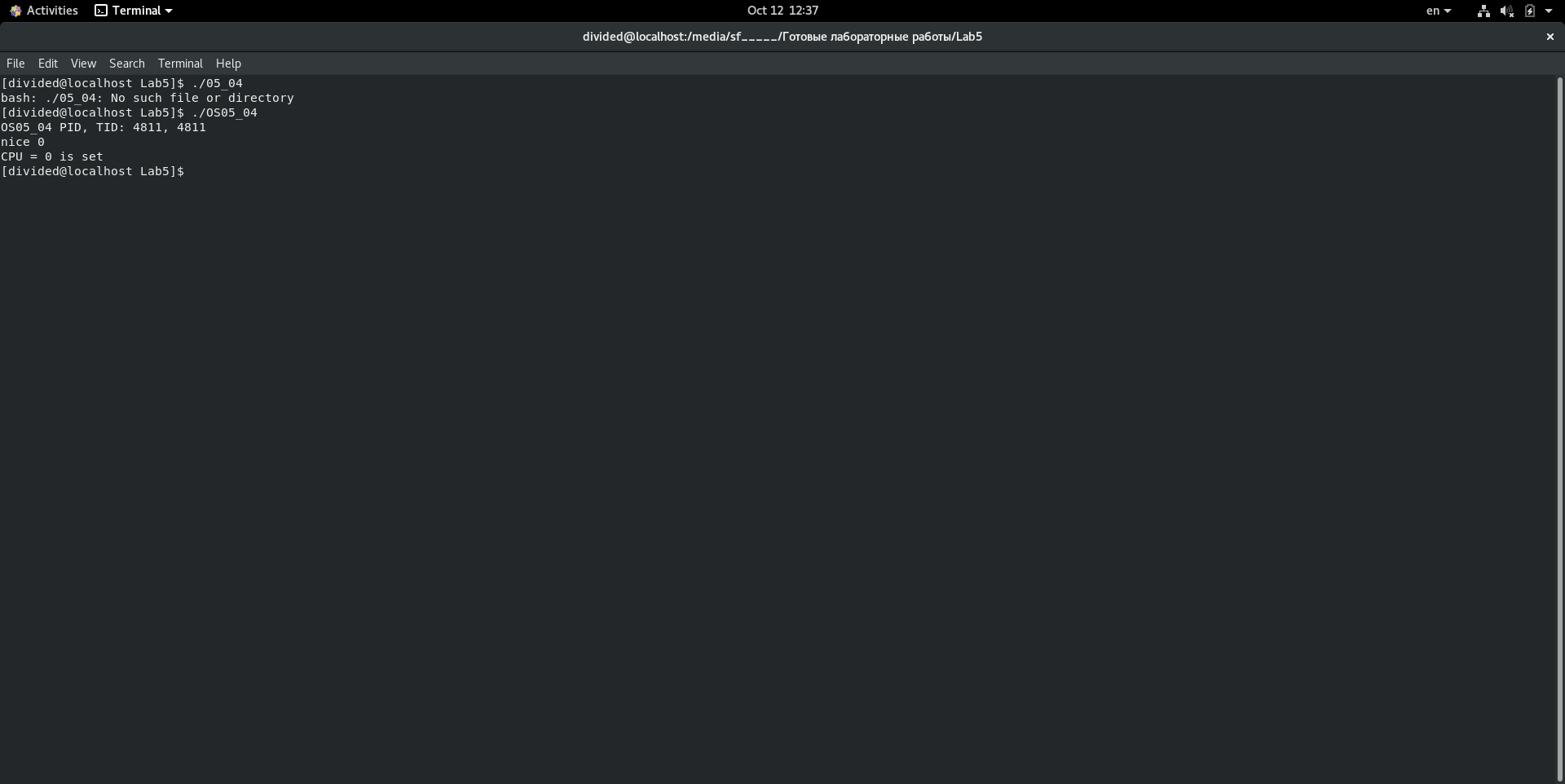
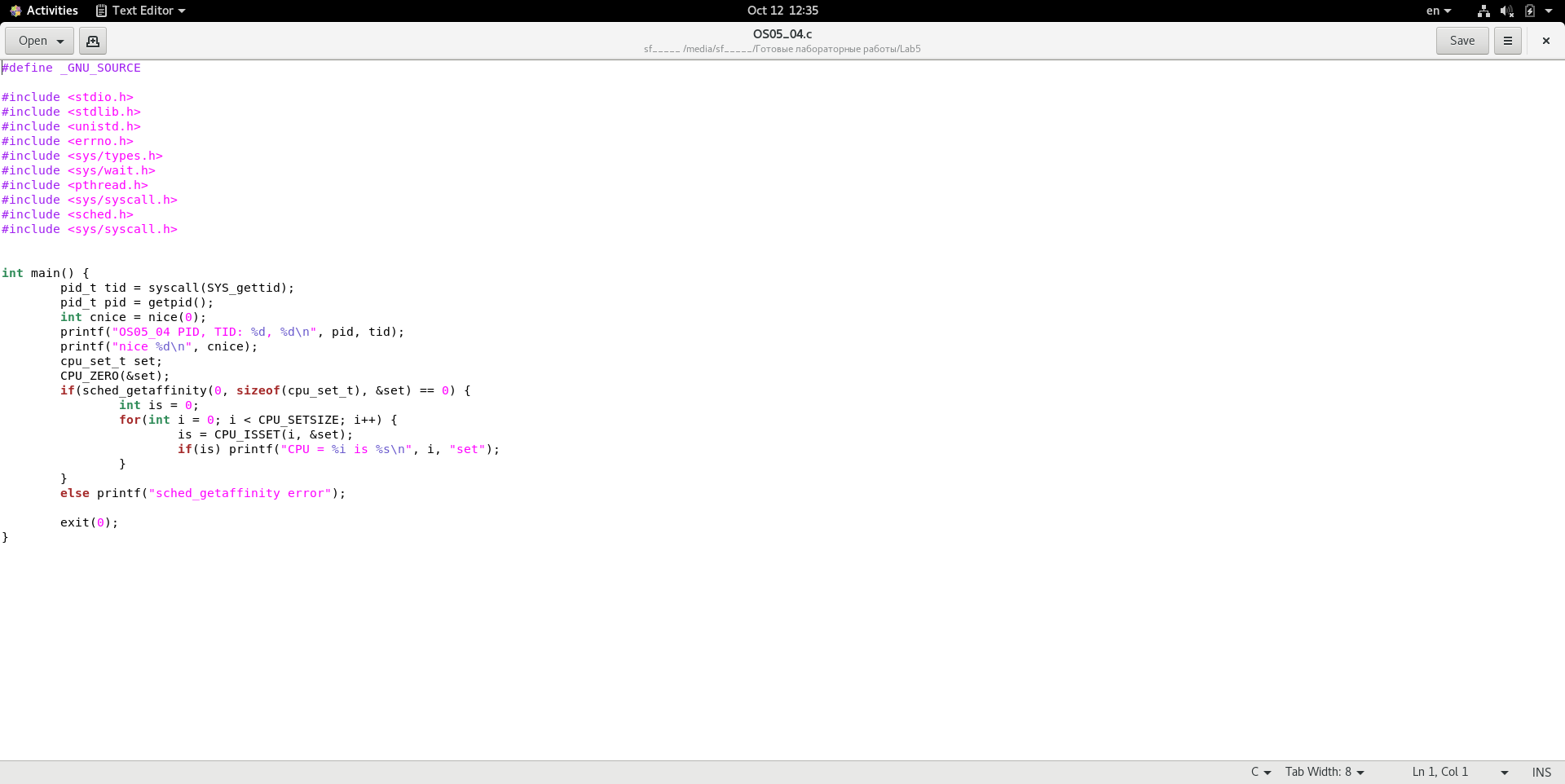
**OS05\_03 255 32 -2 2**



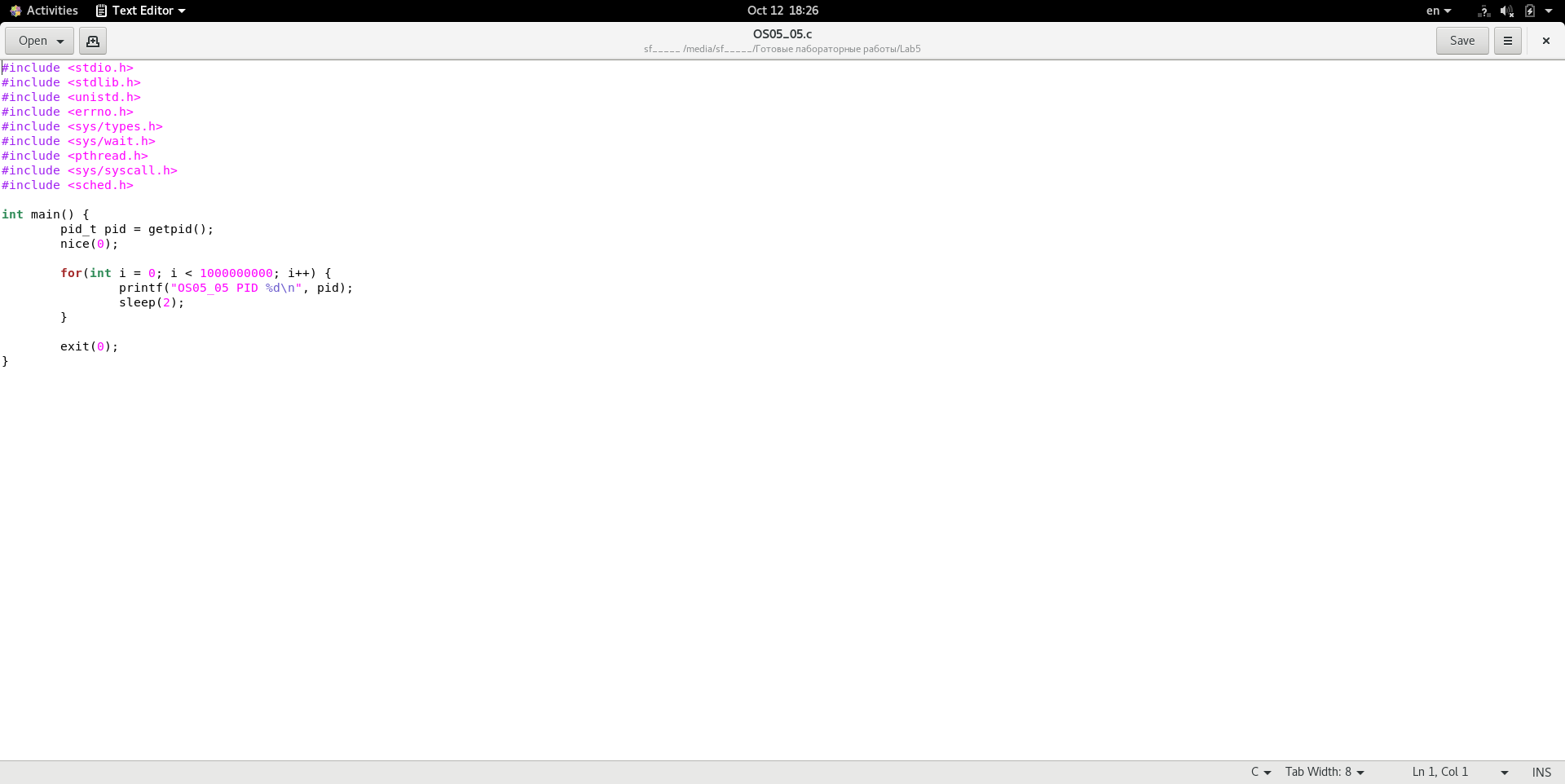
**OS05\_03 1 32 -2 2**

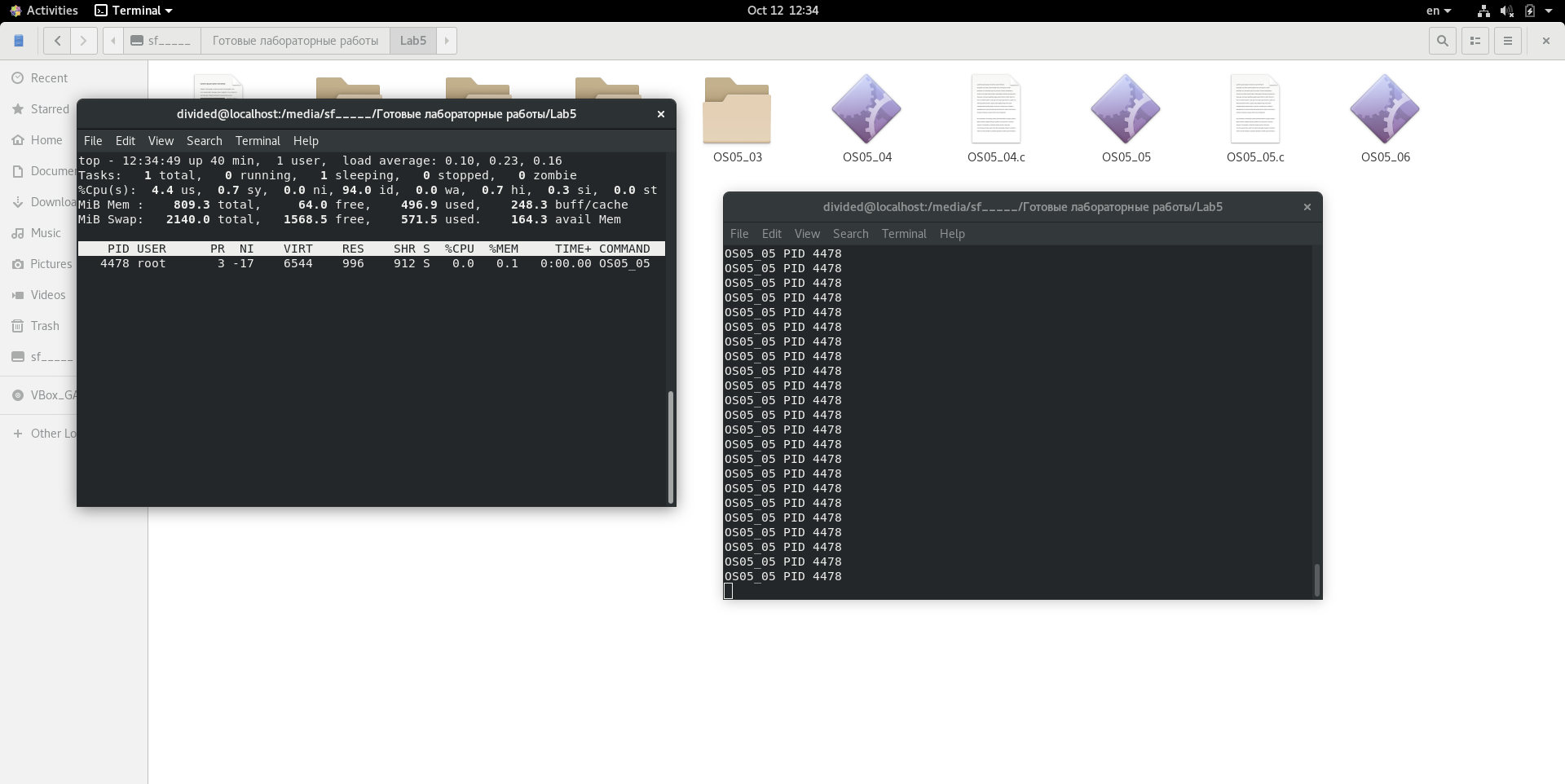
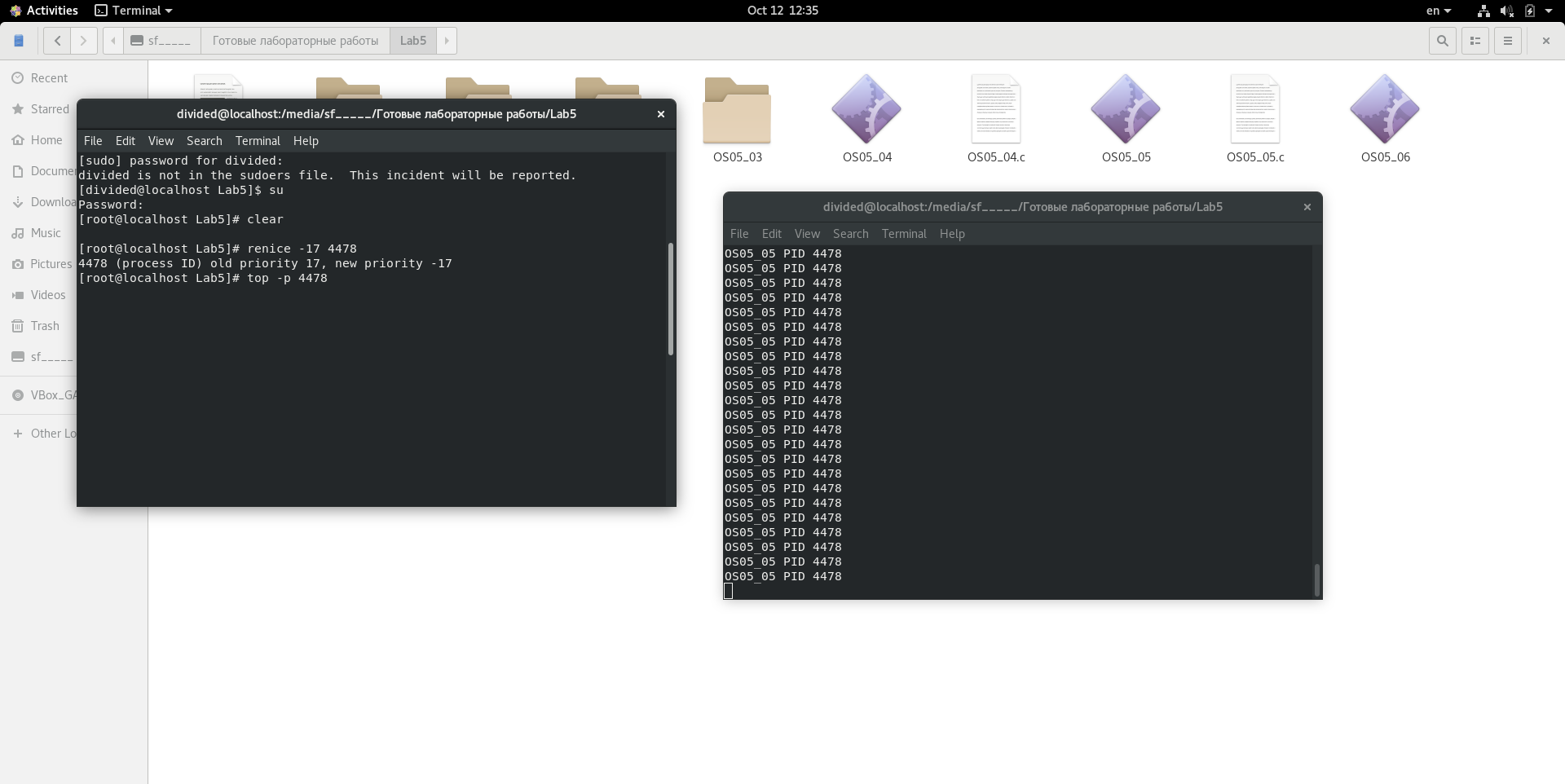
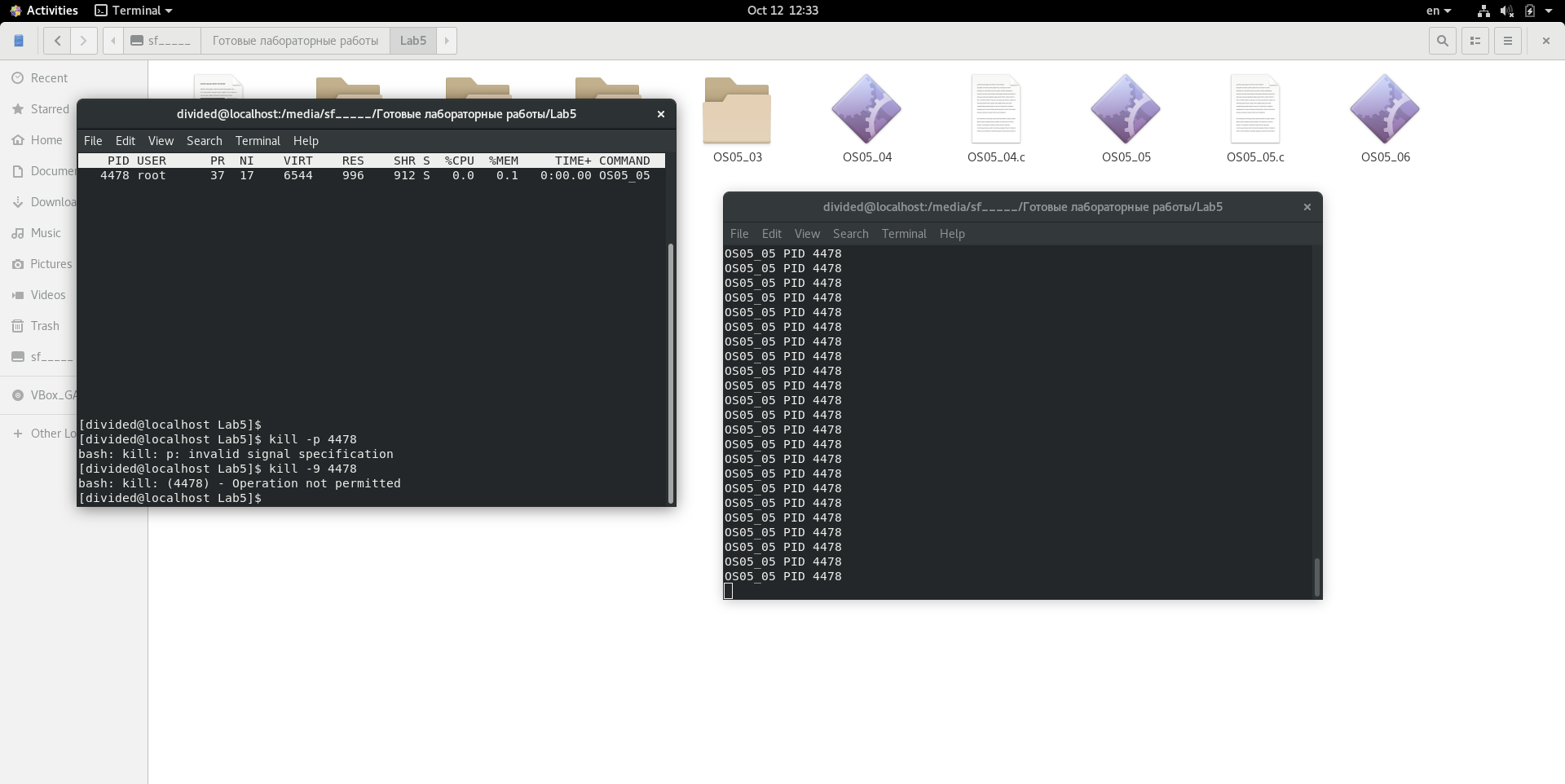
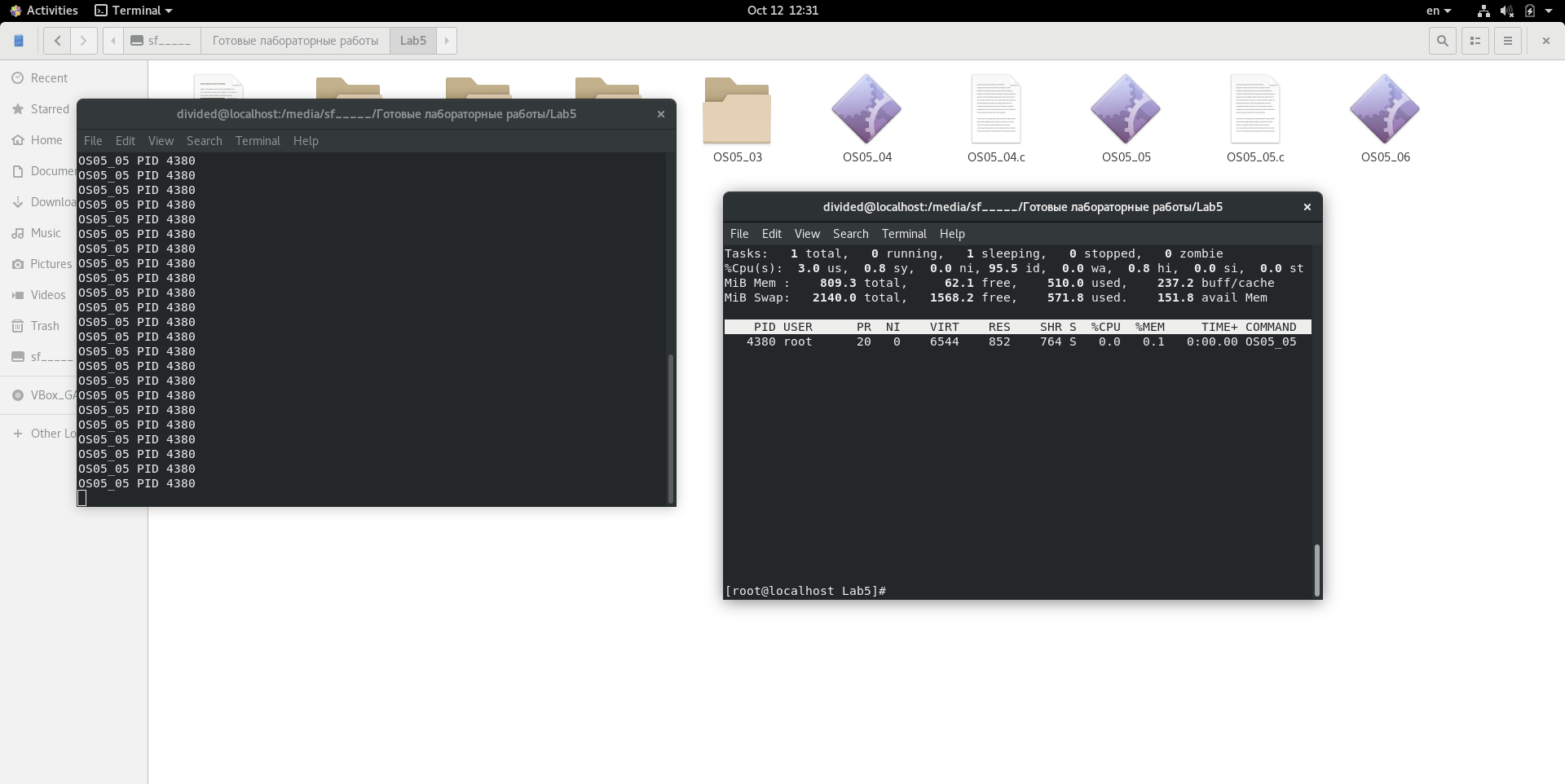


**Задание 4**

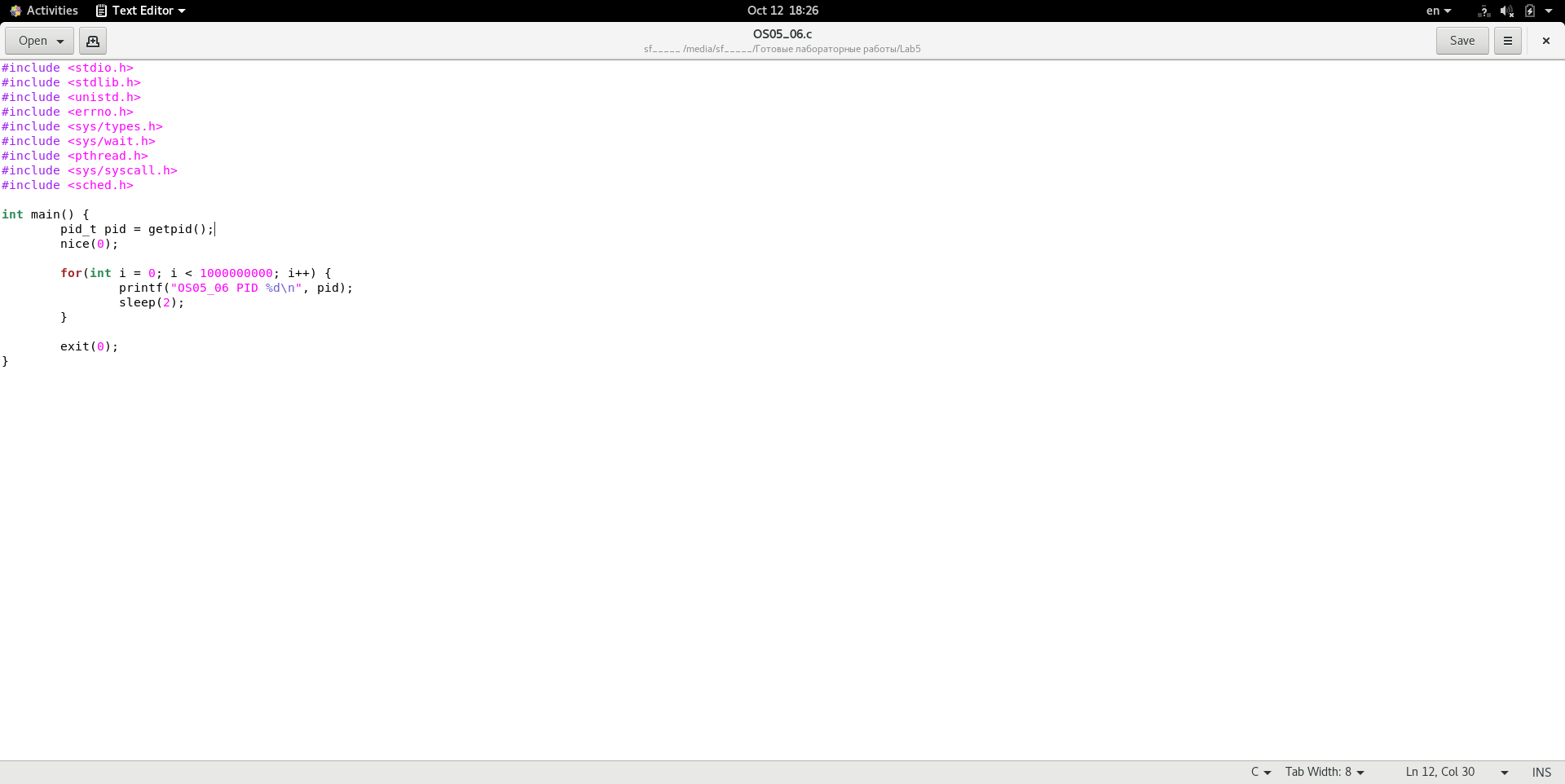


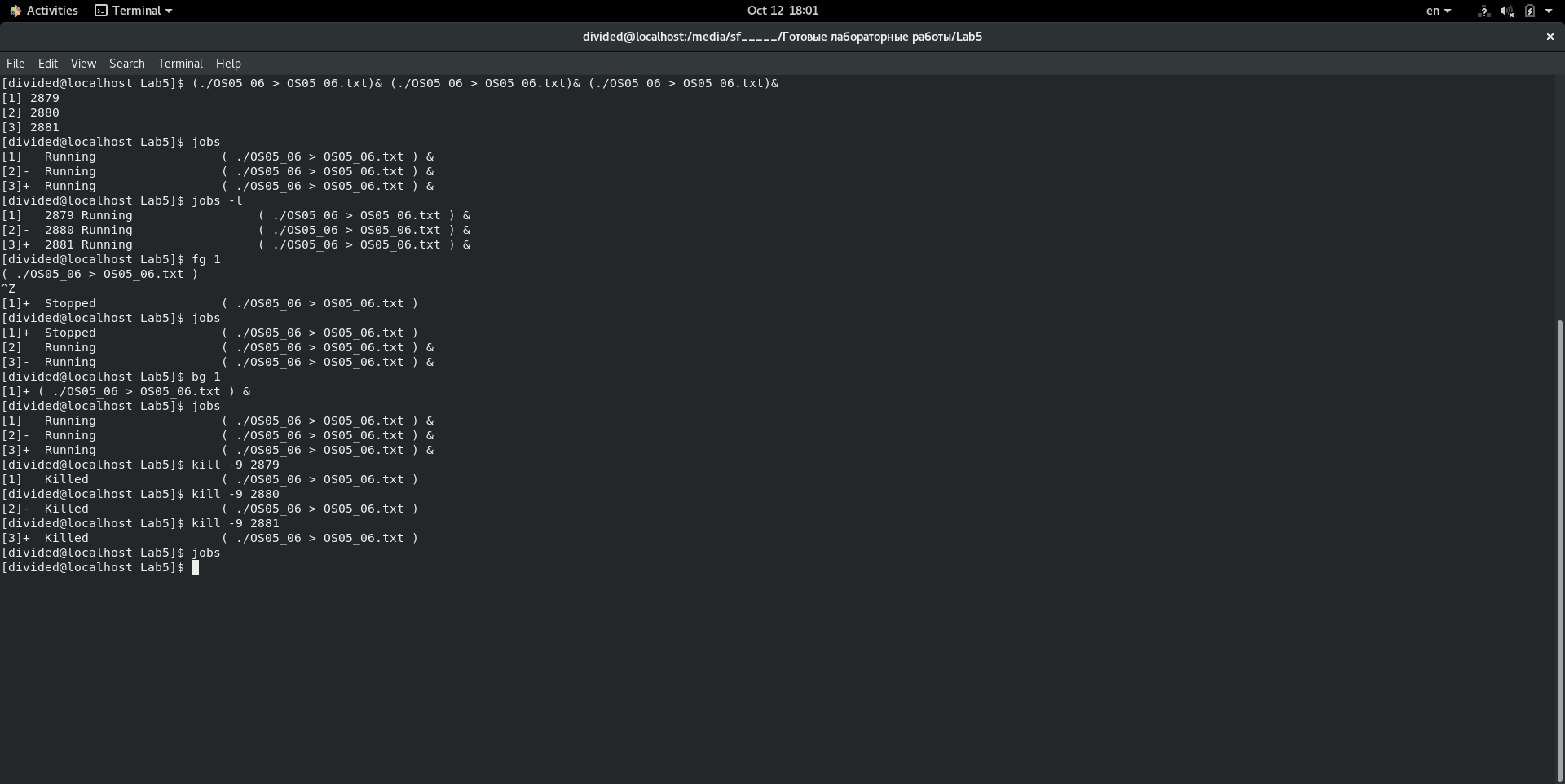
**Задание 5**

****

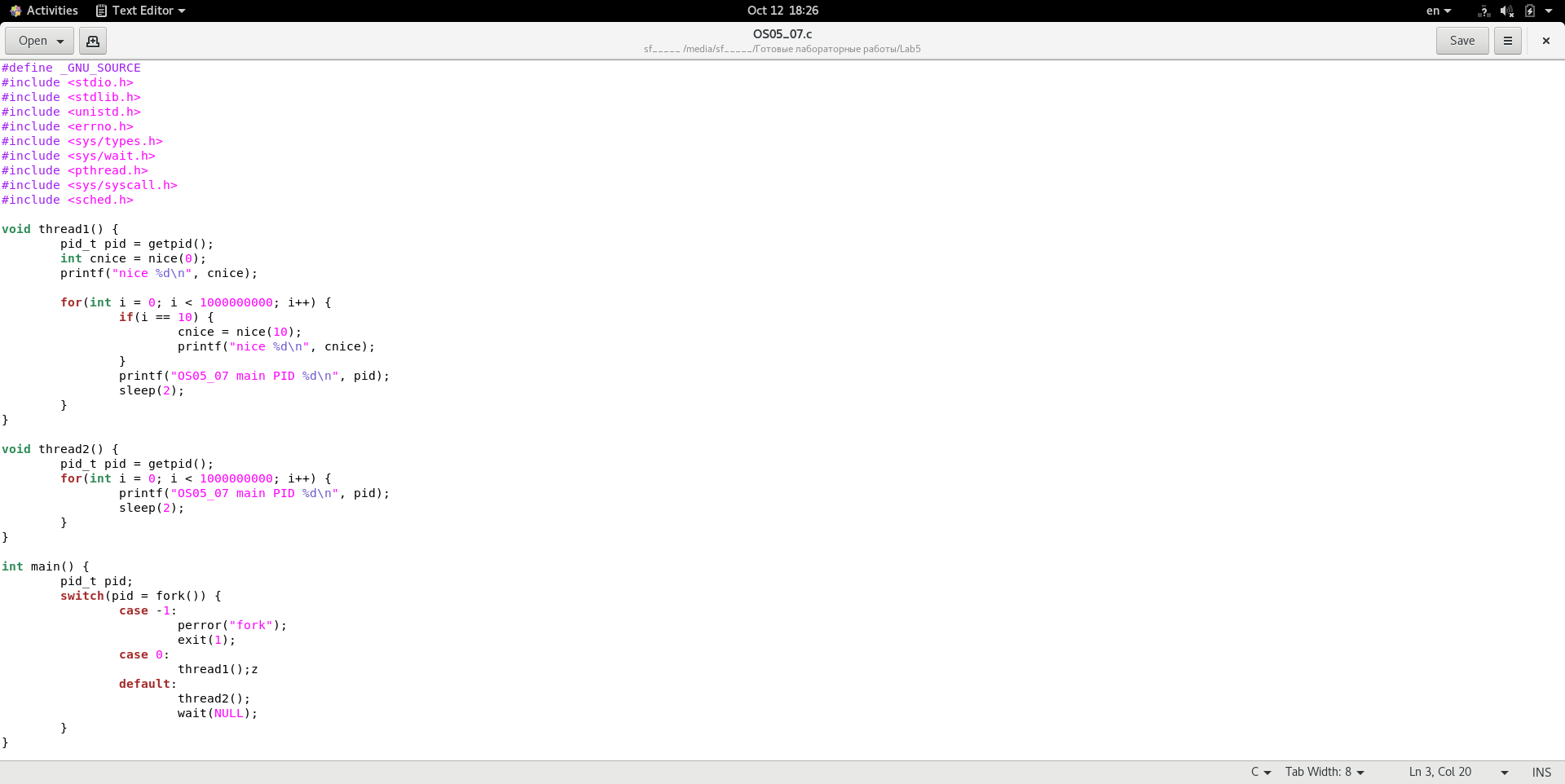


**Задание 6**

****



**Задание 7**

****

